Data Science e Tecnologie per le Basi di Dati

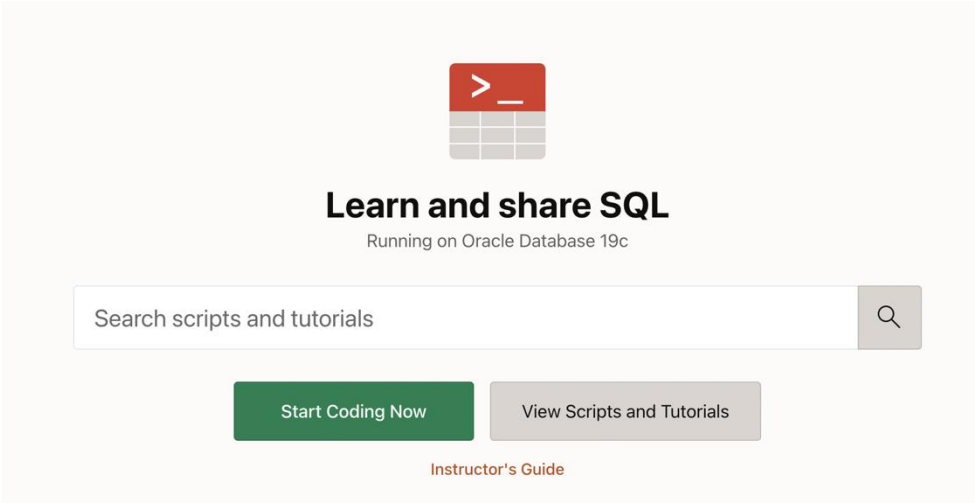
Esercitazione di laboratorio n. 3

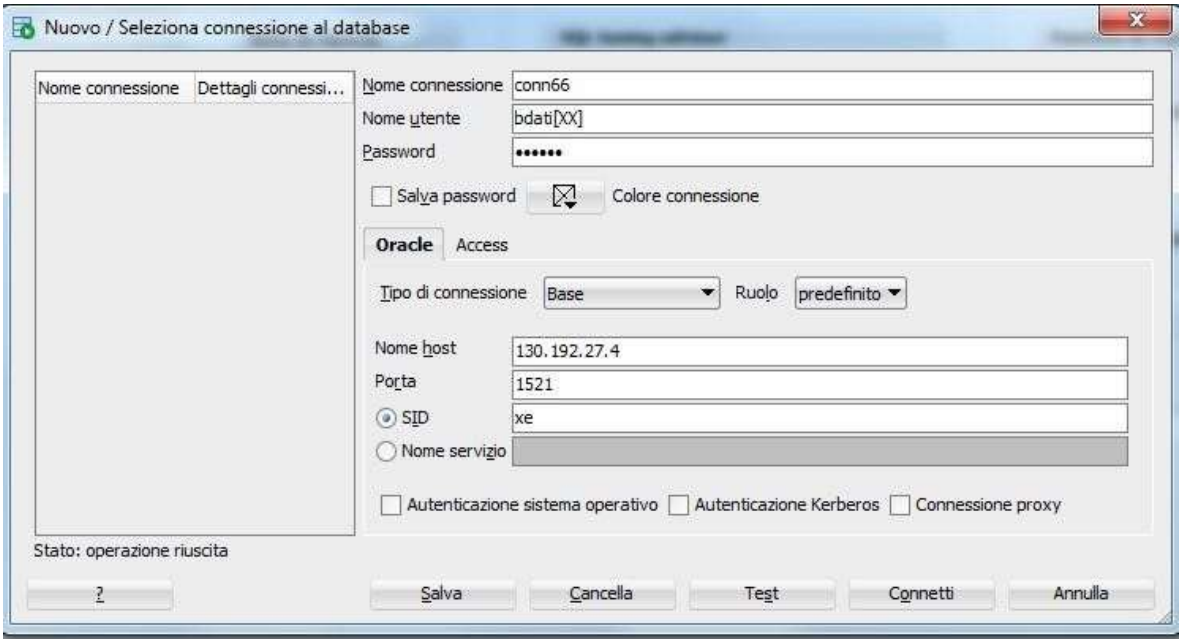
Viste materializzate e trigger

La finalità di questa esercitazione consiste nel definire delle viste materializzate ritenute utili per rispondere velocemente a frequenti interrogazioni del data warehouse.
Le viste create devono poi essere mantenute aggiornate opportunamente, gestendo eventuali modifiche apportate sulle tabelle del data warehouse di partenza.

1. Configurazione

Utilizzare il materiale fornito nella prima esercitazione. Pertanto utizzare Oracle Live SQL e caricare la base dati con i file sql o connettersi alla base dati con SQL developer con la stessa configurazione della prima esercitazione.





2. Creazione e aggiornamento della vista materializzata con l'uso di CREATE MATERIALIZED VIEW in ORACLE

2.1. Creazione e aggiornamento vista materializzata

Esercizio 1.1 Partendo dal data warehouse descritto nella prima esercitazione di laboratorio (e il cui schema logico è riportato nella tabella in Figura 2), definire due viste materializzate utili per ridurre i tempi di risposta di almeno tre delle 6 query riportate di seguito.

Tabelle			**Descrizione**
DWABD.TEMPO (ID_TEMPO DATA GIORNO MESE ANNO **PRIMARY** KEY(ID_TEMPO));	 NUMBER DATE VARCHAR VARCHAR NUMBER	 NOT NULL, NOT NULL, NOT NULL, NOT NULL, NOT NULL,	Dimensione tempo Cardinalità: 30 tuple
DWABD.TARIFFA (ID_TAR TIPO_TARIFFA **PRIMARY** KEY(ID_TAR));	 NUMBER VARCHAR	 NOT NULL, NOT NULL,	Dimensione tariffa Cardinalità: 7 tuple
DWABD.LUOGO (ID_LUOGO CITTA PROVINCIA REGIONE **PRIMARY** KEY(ID_LUOGO));	 NUMBER VARCHAR VARCHAR VARCHAR	 NOT NULL, NOT NULL, NOT NULL, NOT NULL,	Dimensione luogo (località) Cardinalità: 1500 tuple
DWABD.FATTI (ID_TEMPO ID_TAR ID_LUOGO_CHIAMANTE ID_LUOGO_CHIAMATO PREZZO CHIAMATE	 NUMBER NUMBER NUMBER NUMBER NUMBER NUMBER	 NOT NULL, NOT NULL, NOT NULL, NOT NULL, NOT NULL, NOT NULL,	Tabella dei fatti Cardinalità: 7809 tuple
PRIMARY KEY(ID_TEMPO, ID_TAR,ID_LUOGO_CHIAMANTE,ID_LUOGO_CHIAMATO), FOREIGN KEY(ID_TEMPO) REFERENCES TEMPO(ID_TEMPO), FOREIGN KEY(ID_TAR) REFERENCES TARIFFA(ID_TAR), FOREIGN KEY(ID_LUOGO_CHIAMANTE) REFERENCES LUOGO(ID_LUOGO), FOREIGN KEY(ID_LUOGO_CHIAMATO) REFERENCES LUOGO(ID_LUOGO));			

Figura 2 – Tabelle del data warehouse

QUERY SQL:

1. Selezionare l'incasso totale per ogni tipo tariffa e per ogni mese dell'anno 2003. Selezionare inoltre l'incasso totale complessivo, l'incasso totale per ogni tipo di tariffa indifferentemente dal mese, e l'incasso totale per ogni mese indifferentemente dal tipo di tariffa.

2. Per ogni mese, selezionare il numero di chiamate totali e l'incasso totale. Utilizzando la funzione RANK(), associare ad ogni mese un numero che identifica la posizione del mese all'interno dei mesi in funzione dell'incasso totale effettuato (1 per il mese che ha incassato di più, 2 per il secondo mese, ecc.)

3. Selezionare per ogni mese dell'anno 2003 il numero di chiamate totali. Utilizzando la funzione RANK(), associare ad ogni mese un numero che identifica la posizione del mese all'interno dei vari mesi dell'anno 2003 in funzione del numero di chiamate totali (1 per il mese con più telefonate, 2 per il secondo mese, ecc.)

4. Separatamente per tariffa, selezionare l'incasso totale ottenuto nel mese di Luglio dell'anno 2003.

5. Selezionare per ogni mese, l'incasso del mese e l'incasso cumulativo dall'inizio dell'anno.

6. Considerare l'anno 2003. Separatamente per tariffa e mese, analizzare: i) l'incasso totale, ii) la percentuale dell'incasso rispetto all'incasso totale considerando tutte le tariffe telefoniche, iii) la percentuale dell'incasso rispetto all'incasso totale considerando tutti i mesi

Esercizio 1.2 Dopo aver creato le viste materializzate, assicurarsi che queste siano opportunamente aggiornate quando qualche modifica avviene sulla base di dati di partenza. Quali tabelle vanno monitorate per aggiornare di conseguenza le viste create?

STEP A: Provare a modificare il contenuto della tabella FATTI nel seguente modo:

```
insert into fatti(id_tempo, id_tar, id_luogo_chiamante, id_luogo_chiamato, prezzo, chiamate)
values(8,1,558,752,150,40000)

insert into fatti(id_tempo, ID_TAR, id_luogo_chiamante, id_luogo_chiamato, prezzo, chiamate)
values(1,6,558,752,100,100)
```

STEP B: Verificare adesso il nuovo contenuto delle viste materializzate, aggiornandole con il seguente comando:

```
BEGIN
DBMS_SNAPSHOT.REFRESH('Nome_Vista');
END;
```

Come sono cambiate le due viste?

Esercizio 1.3 (Opzionale) Provare a eseguire le query con e senza viste materializzate e verificare che l'output ottenuto sia lo stesso nei due casi.

Esercizio 1.4 (Opzionale, per chi svolge l'esercitazione sul proprio pc) Provare ad aggiornare il contenuto delle viste materializzate, creando i materialized view log per tutte le tabelle necessarie, in modo da monitorare i cambiamenti di ciascuna tabella.

Per quali tabelle è necessario creare le apposite strutture di log?

STEP A: Per ciascuna tabella in cui lo si ritiene necessario, creare il materialized view log nel seguente modo:

```
CREATE MATERIALIZED VIEW LOG ON TABLE1
 WITH ROWID, SEQUENCE(Attribute_1, Attribute_2, Attribute_3)
 INCLUDING NEW VALUES;
```

N.B: I materialized view log vanno creati **prima** della vista materializzata. Eliminare quindi la materialized view nel caso sia stata già creata al punto precedente.

STEP B: Creare la vista materializzata, assicurandosi che l'opzione relativa al metodo di aggiornamento sia settata a "FAST" e che il tipo di aggiornamento sia "ON COMMIT".

STEP C: Modificare la tabella dei fatti come indicato nell'esercizio 1.2 e verificare che la modifica sia correttamente propagata sulle viste materializzate.

N.B: I materialized view log **non** possono essere creati con la versione di Oracle presente sui computer del laboratorio.

3. Aggiornamento e gestione viste tramite Trigger

Supponendo che il comando CREATE MATERIALIZED VIEW non sia disponibile, creare ora le viste materializzate definite nell'esercizio precedente seguendo i passi elencati:

STEP 3.1 Creare la struttura della vista materializzata con la seguente istruzione:

```
CREATE TABLE VM1 (…)
```

STEP 3.2 Popolare la tabella VM1 con i record necessari tramite la seguente istruzione

```
INSERT INTO VM1 (…)
( SELECT …
  … )
```

Esercizio: Si scrivano adesso i Trigger necessari per propagare le modifiche (inserimento di un nuovo record) apportate nella tabella FATTI alle viste materializzate create VM1 e VM2.

Verificare il corretto funzionamento dei trigger eseguendo le seguenti operazioni e verificando che VM1 e VM2 vengano aggiornate di conseguenza:

```
insert into fatti(id_tempo, id_tar, id_luogo_chiamante, id_luogo_chiamato, prezzo, chiamate)
values(8,2,558,752,150, 40000)

insert into fatti(id_tempo, ID_TAR, id_luogo_chiamante, id_luogo_chiamato, prezzo, chiamate)
values(1,7,558,752,100,100)
```

Su quale delle due tabelle è stato eseguito l'aggiornamento di un record già esistente? Su quale invece è stato inserito un nuovo record?

Comandi utili per la gestione dei trigger

- Cancellazione di un trigger:
  - drop trigger nome_trigger;
- Sostituzione (aggiornamento) di un trigger esistente:
  - CREATE OR REPLACE TRIGGER nome_trigger;
- Visualizzazione dei trigger generati:
  - select trigger_name, triggering_event, table_name, status, description, action_type, trigger_body from user_triggers;
- Visualizzazione degli errori dei trigger:
  - select * from USER_ERRORS;

.